ACQUISITION RESEARCH SPONSORED REPORT SERIES

Test and Evaluation of Program Slicing Tools

19 December 2012

by

Dr. Valdis A. Berzins, Professor,

Richard Pudadera, and

Amir Z'ghidi

Graduate School of Operational and Information Sciences

Naval Postgraduate School

			Form Approved OMB No. 0704-0188

Report Documentation Page

1. REPORT DATE **19 DEC 2012**	2. REPORT TYPE	3. DATES COVERED **00-00-2012 to 00-00-2012**

4. TITLE AND SUBTITLE **Test and Evaluation of Program Slicing Tools**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Postgraduate School,Graduate School of Business and Public Policy,Monterey,CA,93943**	8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Software slicing is the analysis of a program based on a slicing criterion to produce executable code that contains only those lines that are related to the slicing criterion. This reduces the state space of the program, thus aiding in testing and debugging. This technical report analyzes the Indus-Kaveri plug-in to assess its capacity for providing correct software slices that can be used to reduce regression tests. The tests reveal that the plug-in produces slices as advertised by the documentation, but, nevertheless, has many limitations that make it impractical for use in a commercial environment. These limitations include the plug-in?s incompatibility with programs that require a Java Runtime later than version 1.4, the lack of highlighting of slices within external classes utilized, and the fact that slicing is accomplished through the return value of a method.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT **Same as Report (SAR)**	18. NUMBER OF PAGES **37**	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**			

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

ACQUISITION RESEARCH PROGRAM
GRADUATE SCHOOL OF BUSINESS & PUBLIC POLICY
NAVAL POSTGRADUATE SCHOOL

Abstract

Software slicing is the analysis of a program based on a slicing criterion to produce executable code that contains only those lines that are related to the slicing criterion. This reduces the state space of the program, thus aiding in testing and debugging. This technical report analyzes the Indus-Kaveri plug-in to assess its capacity for providing correct software slices that can be used to reduce regression tests. The tests reveal that the plug-in produces slices as advertised by the documentation, but, nevertheless, has many limitations that make it impractical for use in a commercial environment. These limitations include the plug-in's incompatibility with programs that require a Java Runtime later than version 1.4, the lack of highlighting of slices within external classes utilized, and the fact that slicing is accomplished through the return value of a method.

Keywords: Software slicing, executable code, slicing criterion, Indus-Kaveri plug-in

THIS PAGE INTENTIONALLY LEFT BLANK

About the Authors

Valdis A. Berzins is a professor of computer science at the Naval Postgraduate School. His research interests include software engineering, software architecture, reliability, computer-aided design, and software evolution. His work includes software testing, reuse, automatic software generation, architecture, requirements, prototyping, re-engineering, specification languages, and engineering databases. Berzins received BS, MS, EE, and PhD degrees from MIT and has been on the faculty at the University of Texas and the University of Minnesota. He has developed several specification languages, software tools for computer-aided software design, and fundamental theory of software merging.

Valdis A. Berzins
Graduate School of Operational & Information Sciences
Naval Postgraduate School
Monterey, CA 93943-5000
Tel: 831-656-2610
Fax: (831) 656-3407
E-mail: berzins@nps.edu

THIS PAGE INTENTIONALLY LEFT BLANK

ACQUISITION RESEARCH
SPONSORED REPORT SERIES

Test and Evaluation of Program Slicing Tools

19 December 2012

by

Dr. Valdis A. Berzins, Professor,

Richard Pudadera, and

Amir Z'ghidi

Graduate School of Operational and Information Sciences

Naval Postgraduate School

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

THIS PAGE INTENTIONALLY LEFT BLANK

I.   Introduction

A.   Slicing Overview

Slicing was introduced in by Mark Weiser (1984) and refers to reducing a program to a smaller form that still produces the behavior of the original program based on a given point of interest called the slicing criterion. Weiser noticed that programmers were actually performing mental slicing when they were debugging their programs and tried to provide a formal definition of this process (De Lucia, 2001).

After Weiser's (1984) introduction of the concept of slicing, much research has been done and many papers have been published. The original definition by Weiser related to static slicing, meaning that the computed slice would always produce the same output as the original program, regardless of the program's input. Variations of the original definition have been introduced, such as dynamic slicing, which identifies the set of statements that affect the variable of interest (slicing criterion) for a particular execution of the program rather than for all possible program executions. Other slicing variations include quasi static slicing, simultaneous dynamic slicing, and conditioned slicing.

A program slice is not unique. In other words, we can find multiple slices that reproduce the same behavior as the original program based on a given slicing criterion. An ideal slice of a program would be a slice that contains the smallest number of statements. However, according to Weiser (1984), finding the minimal slice is not algorithmically solvable, and practical solutions must be content with "small" slices that may not be strictly minimal.

B.   Problem Statement

Even though slicing was introduced a while ago, and despite the benefits it can offer, we do not see many tools that use program slicing to help improve the

software development cycle. In this report, we evaluate a slicing tool and determine whether the tool is suitable for slicing real-world programs.

The first thing we check is whether the tool can produce a correct slice of a given program; otherwise, the tool is useless.

The second thing we analyze is how good the slice generated by the tool is compared to the minimal slice. In fact, the slicing tool would not be as useful as it should be if it included a lot of statements that were irrelevant to the slice.

C.   Existing Tools

A thesis (Lim & Ben Kahia, 2011) was done at the Naval Postgraduate School (NPS) in which the students tried to find a suitable slicing tool. The slicing tools mentioned in this work included CodeSurfer, a C/C++ slicing tool by GrammaTech; Jslice, a static and dynamic slicer for Java programs; and Indus, a static slicing tool for Java programs. The students had licensing issues with CodeSurfer and documentation problems with the Jslice tool, so they decided to analyze Indus.

In this paper, we further analyze the Indus-Kaveri tool to determine whether we can use it with a real-world program. We use test cases to show any limitations the tool might have.

D.   Benefits

Slicing can be very beneficial in code debugging. For example, when a programmer encounters an error in the output of one module, it is much easier to look at the program statements that led to that error rather than scanning the entire code to find the cause of the error. By using a backward slice of the program and choosing the output of the module as the slicing criterion, a perfect slicing tool would only show the statements that affected the output of the module, therefore, making the job of the programmer much easier. Since finding a minimal slice is an undecidable problem, it is likely that some irrelevant statements would be included in

the computed slice; however, this does not deny the fact that debugging a slice of the program requires much less effort than debugging the entire program.

Slicing can also be very useful in software maintenance. Software changes are very common and are needed to improve the software's performance, fix discovered errors, or add new functionality. Making software modifications is very costly because we need to retest the entire system after making changes. This process is referred to as regression testing and is used to validate parts of the software that were modified and, at the same time, ensure that no errors were introduced in the previously tested code. Slicing a program can show us which parts of the program are not affected by the new code and, therefore, we would know which parts can be safely left alone during regression testing. The process of regression testing is a human-intensive activity that makes it costly and time consuming. Therefore, using program slicing to reduce the amount of retesting that needs to be done will help lower the time and cost necessary for software maintenance.

THIS PAGE INTENTIONALLY LEFT BLANK

II.   Tool Analysis

A.   Installation Procedures

Our study found that installation procedures for Indus-Kaveri were poorly documented and difficult to carry out. We found the procedures described in this section to work as of April 9, 2012.

The installation of the Indus-Kaveri plug-in for Eclipse requires the use of legacy versions of Java and Eclipse, as the later versions are not compatible with the legacy plug-ins. The environment used to run Kaveri is as follows:

- Windows 7 (32 bit)

- Eclipse 3.2.0—http://archive.eclipse.org/eclipse/downloads/index.php

- Java 1.4.2_19 (and the VM 5.0)

- Java 1.5.0_22 (optional—plug-in development only)

- Indus Plug-in 0.8.3.14—http://projects.cis.ksu.edu/gf/project/indus/frs/?action=&_br_pkgrls_total=50&_br_pkgrls_page=2

- Kaveri Plug-in 0.8.3—http://projects.cis.ksu.edu/gf/project/indus/frs/?action=&_br_pkgrls_total=50&_br_pkgrls_page=1

- Groovy Monkey 0.6.1—http://sourceforge.net/projects/groovy-monkey/?source=directory

The Indus and Kaveri plug-ins for Eclipse are installed simply by adding the necessary files to the plug-in directory. In the case of Groovy Monkey, the user has to navigate to the "plug-ins" directory of the downloaded .zip file and copy the ".jar" contents to the plug-ins directory of Eclipse. Eclipse and Java have built-in installation/set-up procedures. Note that Eclipse may not run if newer versions of Java have been installed. In such an environment, while the user may get Eclipse to run by using the "-vm" option, the plug-ins will not be available.

Eclipse is stand-alone in that the installed files can simply be copied from one machine to another (i.e., no registry entries or files are installed externally). Once all of these applications and plug-ins have been installed, the slicer has to be configured as follows:

- Open Eclipse and create a new workspace if you don't already have one.

- Click the Indus menu bar item and select "Slicer Configuration."

- On the Configurations tab, select "Create" (see Figure 1).

- Rename the new configuration to something like "my backward slicing."

- In the Slice tab, check the box for "Executable slice" and make sure the slice type is set to "Backward slice."

- Make sure that no other boxes are checked for any of the other tabs.

- Create another configuration and name it something like "my fwd slicing."

- In the Slice tab, check the box for "Property Aware Slicing" and make sure the slice type is set to "Forward slice."

- On the General Dependence tab, check the top three boxes.

- Make sure that no other boxes are checked for any of the other tabs.

- Click "OK"—this has to be done before the next step otherwise the new changes will not be saved.
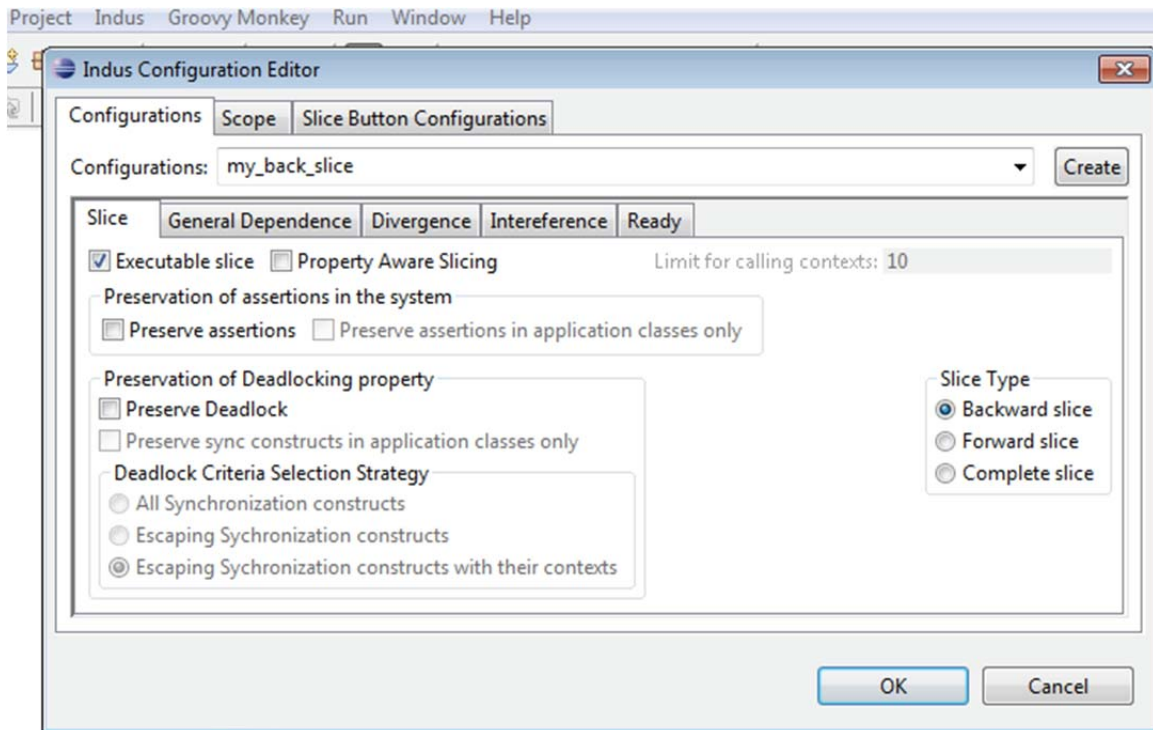
Figure 1. Indus Configuration Editor

Once the two configurations are set up, we need to link them with the buttons:

- Select the "Slice Button Configurations" tab.

- Select your newly created configurations for backward and forward slice actions.

Now you should be able to create a Java file and slice it using the buttons in the Eclipse toolbar (see Figure 2).

Figure 2. Eclipse Toolbar

It should be noted that Java projects are made compliant only with Java 1.4. Errors develop when the project compliance level (see Figure 3) is set to a later version.
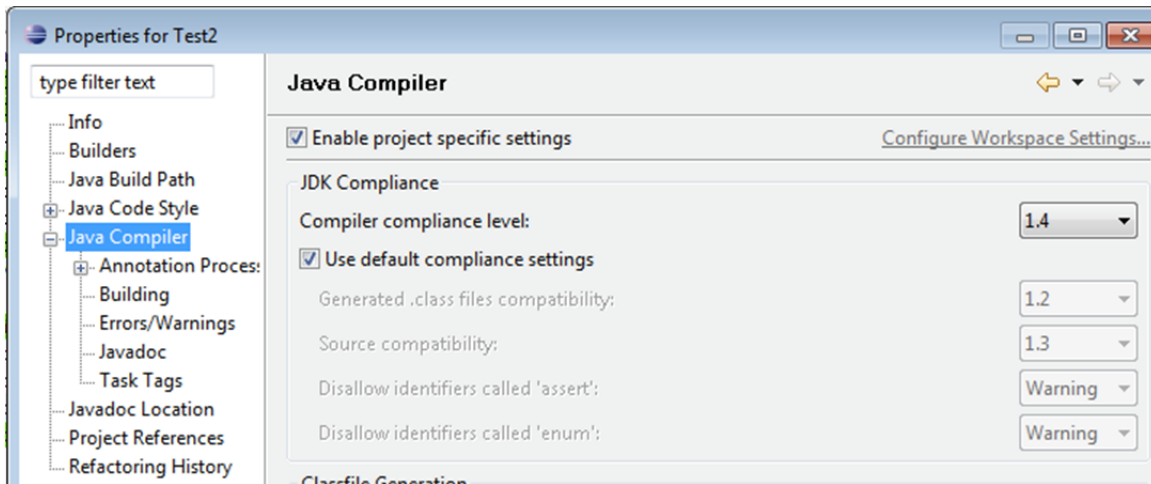
Figure 3. Java Compliance Level

B. Test Cases

After installation, we subjected Kaveri to a series of test cases. The slicing criteria we used were the testing of four simple programs with increasing complexity. The first was a program with all code residing in main. The second was a procedure call within the same class. The third was a call to a separate external class. The fourth class tested Indus compatibility with a recent Java version. The fifth test was an evaluation of the tool's ability to recognize when a class calls itself. The sixth test was an evaluation of the way the tool interprets object references and changes to their respective contents.

C. Test Results and Interpretation

The test results are found in Appendix A. As can be seen from these results, Indus is capable of performing slices, but the results are not perfect. In fact, the slicer sometimes ignores code statements that are essential to the proper functioning of the slice, such as the case with slicing the Test2 file. In addition, when slicing a Java file that calls external classes, Kaveri does not visually provide a way of showing which classes are included in the slice.

Indus appears to work fine with small-scale programs, but many issues start to arise as the complexity of a program grows. Unfortunately, Test 4 revealed a

limitation with Indus in that it only operates using Java 1.4.2. Test 4 incorporates new features available in Java 1.5 and above and, thus, cannot be run using Java 1.4.2. This highlights another limitation with Indus in that it cannot be used with modern Java programs.

Test 5 highlighted an issue that made the tool provide what appears to be an incorrect/incomplete slice. We believe that this behavior is due to Kaveri slicing based on values, specifically, the returned value. To confirm this, Test 5 was modified to return a value. The program was then sliced correctly based on the returned variable. What appeared to be an incorrect slice in the original Test 5 occurred because the *System.out.println()* method does not return a value. This behavior is counterintuitive, as a user would have expected that the slice be performed based on the input variables as well. To allow the program to be sliced in accordance with the way a user expects it to, the program has to be modified to ensure that the variables of interest affect the output object. This can be achieved by manually coding these return values, but preferably through the use of wrappers to automate the work and reduce chances of errors being introduced. Regardless of this additional difficulty, the slicing criterion should not affect the behavior invariance property (i.e., the behavior of the sliced program is no different from the behavior of the original program).

D.   Limitations

- The Indus tool in its current form is unrealistic for real-world scenarios. According to the developer, only programs that are compatible with Java VM 2 can be sliced.

- Kaveri does not appear to automatically slice classes or modules referenced that are not located in the same Java file. This effectively limits the ability of the plug-in to analyze real-world applications. Indus does not identify the end of a code block. This means that the output from Indus is not readily compilable, as it does not specify which lines of code belong to a loop or method.

- Variables must be declared and initialized in the same line. Indus does not appear to consider the declaration of a variable to be relevant to a

slice, and, thus, does not comply with the define-use principle of testing.

- Indus and Kaveri are very difficult to set up and configure, let alone modify. The dependence on external tools and plug-ins that the system relies on is a big hindrance. This is aggravated by the lack of support for the tool, in that the developer is no longer focused on providing updates and maintenance of the software. We performed a web search as well as contacted the person who developed Indus to see if there is a stable commercial version of the tool, and we concluded that there was not any commercial version of Indus-Kaveri.

- As an example of set-up difficulties, the tool requires Java version 1.5 to compile, version 1.4.2 to run, and version 1.2 as a target source. The end result is that the tool is only capable of slicing legacy code.

- Indus ignores the lines with closing brackets. This is significant in "for" and "while" loops, especially if the user wishes for the program to output an executable file.

- Slices are based on value, specifically the return value. If the user wishes to slice based on the input variables as well, then the program has to be modified to return an object based on the input variables.

E.   Tool Alternatives

Some limitations of the Indus-Kaveri tool can be overcome by using the command line version of Indus (SliceXMLizerCLI), along with a parser program that would analyze the output files resulting from slicing a Java program with SliceXMLizerCLI. Unlike the Kaveri plug-in, which does not slice external classes, the output of the SliceXMLizerCLI command would include a slice of external classes called by the program being sliced.

When slicing a Java program P with the Indus command line interface, we would have as an output the Jimple of the slice of program P and Jimple code of other external classes in program P that are needed to compute the slice of P. Jimple is an intermediate representation of a Java program that is easier to understand than Java bytecode, but more difficult to understand than Java code.

III.   Conclusion

The tool analysis provided shows that the Indus-Kaveri plug-in works as described in the product documentation. It does not tend to produce an excessively large slice, thus potentially saving time because it does not run irrelevant code. Despite this, however, it has many limitations that prevent it from being used in a commercial setting. Even if the plug-in were updated to be compatible with Java programs that use the latest Runtime, that display highlighting within external classes used, or that produce complete executable code, it nevertheless would require an impractically major overhaul to allow the program to behave as a user would intuitively expect it to (i.e., slices based on input and output values).
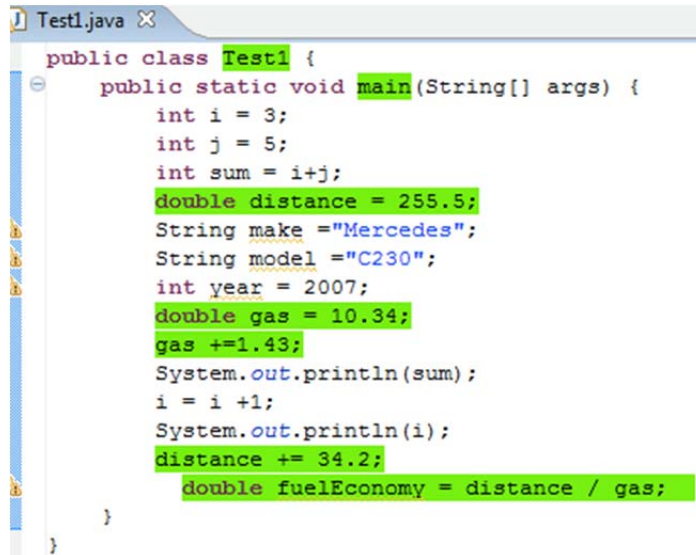
THIS PAGE INTENTIONALLY LEFT BLANK
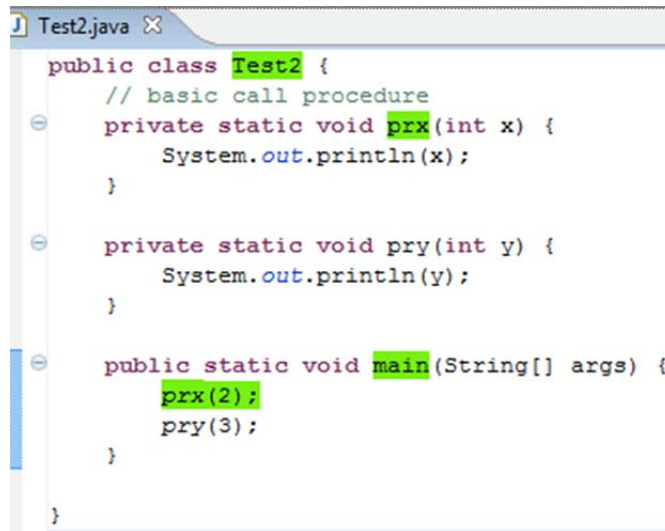
Appendix A. Test Results

A. Test 1 Results

Figure 4. Visual Output for the fuelEconomy Line

The slice highlighted the lines that were expected; no major issues were found.

B.   Test 2 Results

```
J Test2.java ⊠
  public class Test2 {
      // basic call procedure
⊖     private static void prx(int x) {
          System.out.println(x);
      }

⊖     private static void pry(int y) {
          System.out.println(y);
      }

⊖     public static void main(String[] args) {
          prx(2);
          pry(3);
      }

}
```

Figure 5. Visual Output for prx(2)

The slice highlighted the appropriate method to be called, but, interestingly, it did not highlight the *System.out.println(x)* line. More detail on this is found in the main text.

C. Test 3 Results

Selecting for p2.a = 2, the results shown in Figure 6 were obtained.

```
package edu.nps.cs;

public class Test3 {
    public static void main(String[] args) {
        Point p1 = new Point();
        Point p2 = new Point();
        p1.a = 1;
        p2.a = 2;
        if (p1.a == p2.a) {
                    System.out.println("points are equal");
        }
        System.out.println("P1: p1.a = " + p1.a);
        System.out.println("P2: p2.a = " + p2.a);
    }
}
```

Figure 6. Selecting for p2.a = 2

No output or slice was observed for class "Point," but the appropriate lines have been identified.

D. Test 4 Results

The program failed to run due to the use of an autoboxing feature found in Java 1.5 and above. Since Indus and Kaveri work only with Java 1.4.2, Test 4 could not be run or tested.

E. Test 5 Results

```
public class Car {

    public String make = "";

    public String model = "";

    public String year = "";
```

```java
public int mileage = 0;

private int gasConsumption = 0;


public  static void main(String args[]){

    Car c1 = new Car("Mercedes", "C220", "1994");

    Car c2 = new Car("Audi", "A4", "2000");

    c1.setGasConsumption(25);

    c1.mileage=185000;

    c2.setGasConsumption(27);

    c2.mileage=125000;

    c1.displayDetails();

    c2.displayDetails();

}

public Car (String mk, String md, String yr){

    this.make=mk;

    this.model=md;

    this.year=yr;

    this.mileage=0;

}



public int getGasConsumption() {

    return this.gasConsumption;
```

```java
        }

        public void setGasConsumption(int gasConsumption) {

            this.gasConsumption = gasConsumption;

        }

        public  void displayDetails(){

            System.out.println("Make: " + this.make);

            System.out.println("Model: " + this.model);

            System.out.println("Year: " + this.year);

            System.out.println("Mileage: " + this.mileage);

            System.out.println("");

        }

    }
```

Based on the results of Test 5, it appears that the tool provides an incorrect slice as the line `c1.mileage=185000;` is not highlighted when it should be.

As a further test, the program was slightly modified to provide an output based on the displayDetails method:

```java
package nps.edu;



public class Car {

    public String make = "";

    public String model = "";
```

```java
public String year = "";

public int mileage = 0;

private int gasConsumption = 0;



public  static void main(String args[]){

    Car c1 = new Car("Mercedes", "C220", "1994");

    Car c2 = new Car("Audi", "A4", "2000");

    c1.setGasConsumption(25);

    c1.mileage=c1.mileage + 1;

    c2.setGasConsumption(27);

    c2.mileage=125000;

    int x;

    x = c1.displayDetails();

    System.out.println(x);

    //c2.displayDetails();

}

public Car (String mk, String md, String yr){

    this.make=mk;

    this.model=md;

    this.year=yr;

    this.mileage=0;

}
```

```java
public int getGasConsumption() {

    return this.gasConsumption;

}

public void setGasConsumption(int gasConsumption) {

    this.gasConsumption = gasConsumption;

}

public  int displayDetails(){

    System.out.println("Make: " + this.make);

    System.out.println("Model: " + this.model);

    System.out.println("Year: " + this.year);

    System.out.println("Mileage: " + this.mileage);

    return this.mileage;

}

}
```

Having the program return a value related to mileage causes the tool to highlight the appropriate lines.

F.   Test 6 Results

```java
package edu.nps.cs;



public class Test3 {

    public static void main(String[] args) {
```

```java
Point p1 = new Point();

Point p2 = new Point();

p1.a = 1;

p2 = p1;

if (p1.a == p2.a) {

        System.out.println("points are equal");

}

p2.a = 2;

if (p1.a == p2.a) {

System.out.println("points are still equal");

} else {

        System.out.println("points are no longer equal");

}

p2.a = p2.a + 1;

System.out.println("P1: p1.a = " + p1.a);

System.out.println("P2: p2.a = " + p2.a);

    }

}
```

Apart from the missing declaration of the point p2, the slice is otherwise correct. The Indus-Kaveri plug-in detects the relationship between two objects that have been equated to each other.

List of References

De Lucia, A. (2001). Program slicing: Methods and applications. *Proceedings of the First IEEE International Workshop on Source Code Analysis and Manipulation* (pp.142–149). doi:10.1109/SCAM.2001.972660

Lim, P., & Ben Kahia, M. (2011). *Suitability of commercial slicing tools for safe reduction of testing effort* (Unpublished master's thesis). Naval Postgraduate School, Monterey, CA.

Weiser, M. (1984). Program slicing. *IEEE Transactions on Software Engineering*, *SE-10*(4), 352–357.

THIS PAGE INTENTIONALLY LEFT BLANK

2003 - 2012 Sponsored Research Topics

Acquisition Management

- Acquiring Combat Capability via Public-Private Partnerships (PPPs)
- BCA: Contractor vs. Organic Growth
- Defense Industry Consolidation
- EU-US Defense Industrial Relationships
- Knowledge Value Added (KVA) + Real Options (RO) Applied to Shipyard Planning Processes
- Managing the Services Supply Chain
- MOSA Contracting Implications
- Portfolio Optimization via KVA + RO
- Private Military Sector
- Software Requirements for OA
- Spiral Development
- Strategy for Defense Acquisition Research
- The Software, Hardware Asset Reuse Enterprise (SHARE) repository

Contract Management

- Commodity Sourcing Strategies
- Contracting Government Procurement Functions
- Contractors in 21st-century Combat Zone
- Joint Contingency Contracting
- Model for Optimizing Contingency Contracting, Planning and Execution
- Navy Contract Writing Guide
- Past Performance in Source Selection
- Strategic Contingency Contracting
- Transforming DoD Contract Closeout
- USAF Energy Savings Performance Contracts
- USAF IT Commodity Council
- USMC Contingency Contracting

Financial Management

- Acquisitions via Leasing: MPS case
- Budget Scoring
- Budgeting for Capabilities-based Planning
- Capital Budgeting for the DoD
- Energy Saving Contracts/DoD Mobile Assets
- Financing DoD Budget via PPPs
- Lessons from Private Sector Capital Budgeting for DoD Acquisition Budgeting Reform
- PPPs and Government Financing
- ROI of Information Warfare Systems
- Special Termination Liability in MDAPs
- Strategic Sourcing
- Transaction Cost Economics (TCE) to Improve Cost Estimates

Human Resources

- Indefinite Reenlistment
- Individual Augmentation
- Learning Management Systems
- Moral Conduct Waivers and First-term Attrition
- Retention
- The Navy's Selective Reenlistment Bonus (SRB) Management System
- Tuition Assistance

Logistics Management

- Analysis of LAV Depot Maintenance
- Army LOG MOD
- ASDS Product Support Analysis
- Cold-chain Logistics
- Contractors Supporting Military Operations
- Diffusion/Variability on Vendor Performance Evaluation
- Evolutionary Acquisition
- Lean Six Sigma to Reduce Costs and Improve Readiness

- Naval Aviation Maintenance and Process Improvement (2)
- Optimizing CIWS Lifecycle Support (LCS)
- Outsourcing the Pearl Harbor MK-48 Intermediate Maintenance Activity
- Pallet Management System
- PBL (4)
- Privatization-NOSL/NAWCI
- RFID (6)
- Risk Analysis for Performance-based Logistics
- R-TOC AEGIS Microwave Power Tubes
- Sense-and-Respond Logistics Network
- Strategic Sourcing

Program Management

- Building Collaborative Capacity
- Business Process Reengineering (BPR) for LCS Mission Module Acquisition
- Collaborative IT Tools Leveraging Competence
- Contractor vs. Organic Support
- Knowledge, Responsibilities and Decision Rights in MDAPs
- KVA Applied to AEGIS and SSDS
- Managing the Service Supply Chain
- Measuring Uncertainty in Earned Value
- Organizational Modeling and Simulation
- Public-Private Partnership
- Terminating Your Own Program
- Utilizing Collaborative and Three-dimensional Imaging Technology

A complete listing and electronic copies of published research are available on our website: www.acquisitionresearch.net

ACQUISITION RESEARCH PROGRAM
GRADUATE SCHOOL OF BUSINESS & PUBLIC POLICY
NAVAL POSTGRADUATE SCHOOL

THIS PAGE INTENTIONALLY LEFT BLANK